Correctly Rounded Exponential Function in Double Precision Arithmetic

David Defour, Florent De Dinechin, Jean-Michel Muller

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Correctly Rounded Exponential Function in Double Precision Arithmetic

David Defour — Florent de Dinechin — Jean-Michel Muller

N° 4231

Juillet 2001

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4231--FR+ENG

Correctly Rounded Exponential Function in Double Precision Arithmetic

David Defour , Florent de Dinechin , Jean-Michel Muller

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 4231 — Juillet 2001 — 30 pages

Abstract: We present an algorithm for implementing *correctly rounded* exponentials in double-precision floating point arithmetic. This algorithm is based on floating-point operations in the widespread IEEE-754 standard, and is therefore more efficient than those using multiprecision arithmetic, while being fully portable. It requires a table of reasonable size and IEEE-754 double precision multiplications and additions. In a preliminary implementation, the overhead due to correct rounding is a 2.3 times slowdown when compared to the standard library function.

Key-words: exponential, correct rounding, double-precision.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 —Télécopie +33 4 76 61 52 52

Exponentielle avec arrondi correct en arithmétique flottante double précision

Résumé : Nous présentons un algorithme de calcul d'exponentielle en double précision avec *arrondi correct*. Cet algorithme est basé sur les opérations définies par la norme IEEE-754, plus rapides que l'usage des méthodes faisant intervenir les bibliothèques multiprécisions. Il utilise une table de taille raisonnable ainsi que les additions et multiplications définies par la norme IEEE-754. Dans une version préliminaire nous obtenons le calcul de l'exponentielle avec arrondi correct en 2.3 fois plus de temps que par la fonction fournie par la librairie mathématique standard.

Mots-clés : exponentielle, arrondi correct, double-précision.

1 Introduction

The need for accurate elementary functions is important in many critical programs. Many existing methods are available for computing exponentials, most notably table-based methods[7, 18, 19, 20], polynomial approximations and mixed methods[5]. See the book by Muller [13] for a recent survey on the subject. In this paper we introduce a mixture of Wong and Goto [21] and Tang [18] methods. We hope to describe the method with sufficient implementation details to allow an easy duplication.

The main advantage of our method over those previously cited is that we ensure *correct rounding*: the result returned is the floating-point number closest to the exact mathematical value. To achieve this, we use Vincent Lefèvre's recent work on the Table Maker's Dilemma [11, 12], in order to know which precision will suffice to provide correct rounding in double precision.

One of our major goals in this paper was portability, which explains that we only use the IEEE-754 double-precision format and operations. In particular we don't use specific hardware like fused multiply-and-add units [9, 17], nor double-extended precision.

This paper is organized as follows: next section presents a detailed overview of our method. The sections 3 to 6 describe the method step by step. Section 7 sums up the algorithm and compares it to other known implementations. Section 8 concludes and describes future work.

2 Overview of the method

2.1 Correct rounding and the Table Maker's Dilemma

With the exception of zero, the exponential of a floating-point number is a transcendental number, and can therefore not be represented exactly in standard numeration systems. The only hope is to compute the floating-point number that is closest to such an exponential (which we call the correctly rounded exponential). To achieve this, it is possible to compute an approximation to an exponential whith arbitrary precision. Sometimes, when the exponential is very close to the middle of two consecutive floating-point numbers, the precision required to decide the rounding may be much larger than the precision of the target format. This is known as the Table Maker's Dilemma (TMD).

A method described by Ziv [22] is to increase the precision of the approximation until the correctly rounded value can be decided. Although this iteration is proven to terminate, there was until recently no practical bound on the termination time, which might prevent using this method in critical application. To improve on this, Vincent Lefèvre [11, 12] computed, over the whole double-precision range, the worst cases of extra accuracy that are needed for rounding exponentials correctly. Thanks to this work, we are able to guarantee correct rounding in two iterations only, which we may then optimize separately. The first of these iterations is relatively fast and provides 73 bits of accuracy, which is sufficient in most cases. It will be referred throughout the paper as the **Quick** phase of the algorithm. The second

phase, referred to as the **Accurate** phase, is dedicated to challenging cases. It is slower (although with reasonably bounded execution time), but occurs much more rarely.

2.2 Underlying operations

There are mainly two possibilities to implement correctly rounded functions. The first one uses the integer hardware of the processor to implement some digit-based multiprecision algorithm, suitably tailored for the required accuracy [3, 1, 2].

We chose the second possibility, which is to use the floating-point hardware which is very optimized in current processors. We then have to work with a fixed accuracy: double precision floating-point numbers in the IEEE-754 standard (implemented by most current processors) have 53 bits of mantissa. This standard also very precisely defines the behaviour for the basic operations, which will ensure the portability of our implementation. It will also allow us to prove error bounds for our algorithms.

The problem is then to represent intermediate variables with an accuracy greater than that of the IEEE-754 numbers. We use the solution sometimes referred as *floating-point expansions* [14, 15, 4] : A high-precision number is represented as the sum of several ordinary floating-point numbers.

2.3 Performance

Concerning performance, our goal was to

- *always* provide correctly rounded results,
- keep the *average* computation time low,
- bound the *worst case* computation time with a sensible value, allowing for an use in critical applications.

An intuitive probabilistic argument is that for any extra bit computed, the number of cases for which we are unable to decide the rounding is halved. There is therefore a tradeoff between accuracy (percentage of correct rounding) and performance. This tradeoff is already the spirit of Ziv's method [22]. It lead us to chose the decomposition between the **Quick** and **Accurate** phases.

Lefèvre showed that there were cases when up to 126 bits of accuracy were needed to decide correct rounding to nearest for the exponential. However he also found that there were very few cases (a few dozens over the whole floating-point range) which needed more than 110 bits. We found a method accurate to 110 bits which is much faster than the methods we could find for 126 bits of accuracy, therefore we chose to tabulate the cases where 110 bits are not enough. This improves the average time and adds very little to the worst case time.

2.4 Outline of the method

With the previous considerations, our algorithm will be:

- **Quick** phase: Quickly compute an approximation accurate to 73 bits of the exponential. This accuracy is, simply speaking, the best tradeoff we found, using IEEE-754 double precision, between extra accuracy and computation time.
- If this 73-bit approximation is enough to decide the rounding to an IEEE-754 double, then return the correctly rounded result.
- **Accurate** phase otherwise (which according to probabilistic arguments is the case once out of $2^{(73-53)} \approx 1$ million): compute an approximation to 110 bits of precision.
- If this 110-bit approximation is enough to decide the rounding, then return the correctly rounded result, otherwise look up the result in Lefèvre's worst case table.

The computation of both approximations is classically split into three steps:

1. a *range reduction* that replaces the input number x with a number t which lies in a much smaller interval, while setting up data that will allow to reconstruct e^x out of e^t,
2. a *polynomial approximation* to e^t which is accurate enough on this small interval, and
3. a *reconstruction* which computes e^x out of e^t, using the data set up in the first part.

For each of these three steps, the method used in the **Quick** and **Accurate** phases of the algorithm will be very similar, therefore we have chosen to discuss both methods simultaneously. However the reader should keep in mind that the **Accurate** phase occurs rarely.

2.5 Notations and useful results

Throughout the paper, we will note $+$, $-$ and $\times$ the usual mathematical operations, and $\oplus$, $\ominus$ and $\otimes$ the corresponding floating-point operations in IEEE-754 double precision, in the IEEE-754 *round to nearest* mode.

For a floating-point number x, we will classically denote $\mathrm{ulp}(x)$ the value of the least significant bit of its mantissa.

In the rest of the paper we will use δ to refer to the error commited by the **Quick** phase of the algorithm and ϵ to refer to the error commited by the **Accurate** phase of the algorithm.

We will make use of the following well-known results :

Theorem 2.1 (Sterbenz Lemma [16, 8]) *If x and y are floating-point numbers, and if $y/2 \leq x \leq 2y$ then $x \ominus y$ is computed exactly, without any rounding error.*

Theorem 2.2 (Fast2sum algorithm [10]) *If a and b be floating-point numbers, then the following method computes two floating-point numbers s and r, such that $s + r = a + b$ exactly, and s is the floating-point number which is closest to $a + b$.*

$$
\begin{array}{l}
(s,\ r) = Fast2sum(a,\ b) \\
\quad s = a \oplus b \\
\quad if\ (a > b)\ then \qquad z = s \ominus a \\
\qquad r = b \ominus z \\
\quad else \qquad z = s \ominus b \\
\qquad r = a \ominus z
\end{array}
$$

We can consider a test as an addition in term of execution time. Hence, 4 additions are needed to have the exact sum of two floating point numbers.

Theorem 2.3 (double multiplication[6, 10]) *Let U and V be two floating-point numbers, with $p \geq 2$ the size of their mantissa. Let $c = 2^{\frac{\lceil p \rceil}{2}} + 1$. The following method computes the two floating-point numbers W and W' such that $U \times V = W + W'$:*

$$
\begin{array}{l}
(W,\ W') = Dekker(U,\ V) \\
if\ U > 2^{970}\ then\ u = U \times 2^{-53}\ else\ u = U \\
if\ V > 2^{970}\ then\ v = V \times 2^{-53}\ else\ v = V \\
\quad u' = u \otimes c, \qquad\qquad v' = v \otimes c \\
\quad u_1 = (u \ominus u') \oplus u', \quad v_1 = (v \ominus v') \oplus v' \\
\quad u_2 = u \ominus u_1, \qquad\qquad v_2 = v \ominus v_1 \\
\quad W = u \otimes v \\
\quad W' = (((u_1 \otimes v_1 \ominus W) \oplus (u_1 \otimes v_2)) \oplus (u_2 \otimes v_1)) \oplus (u_2 \otimes v_2) \\
if\ U > 2^{970}\ then\ W = W \times 2^{53} \\
if\ V > 2^{970}\ then\ W' = W' \times 2^{53}
\end{array}
$$

We have to test U and V before and after the core of the algorithms in order to avoid overflow by multiplying by c. If we approximate the cost of this test to 1 addition, the global cost in the worst case is 14 additions and 11 multiplications. If we are sure that U and V are less then 2^{970} we can skip this test, which reduces the cost of this algorithm to 10 additions and 7 multiplications.

3 Range reduction

3.1 Overview of the range reduction

The typical floating-point range reduction for the exponential replaces the input floating-point number x with a smaller number t and an integer N such that we have a *range reduction equation* of the form $e^x = 2^N \times e^t$. After evaluation of e^t, the reconstruction is a product without error thanks to radix-2 arithmetic.

Our problem is that we want the accuracy of the result to exceed that of double-precision floating-point numbers. The purpose of this section is therefore to present a succession of range reduction equations that allows a reconstruction accuracy greater than IEEE-754 double precision.

These reductions are based on the following idea: We first consider an approximation to e^x as a floating-point number y. We then define $\ell = \ln y$, which is not representable as a floating-point number, and we write the exact equation $e^x = e^x \times y \times e^{-\ell}$. We now need to approximate ℓ so that the previous equation holds for the required precision. As mentionned earlier, we do this by writing ℓ in the form of an expansion, say $\ell = \ell^h + \ell^l$. The previous equation becomes $e^x = e^{x-\ell^h} \times y \times e^{-\ell^\ell}$. Here, our reduced argument will be $x - \ell^h$, which is indeed smaller than x since $e^{\ell^h} \approx e^\ell \approx e^x$. In addition we will ensure that since $\ell^h \approx x$, $x - \ell^h$ will be a FP number and can be exactly computed thanks to Sterbenz lemma.

The difficulty is, of course, to select y knowing x. The idea developped here is to read y (along with the expansion of its logarithm ℓ) in a table addressed by the most significant bits of x. To keep this table small enough, however, we need to perform several consecutive steps of such a range reduction. Reconstructing from each step will mean to add expansions, which is much less expensive than to multiply them. Our method involves four such steps, which are detailed in the following, along with the computation of the useful error bounds.

3.2 Overflows and underflows

In the following we will consider input numbers in the range $[A, B]$ where A and B are respectively the logarithms of the smallest and largest representable IEEE-754 double precision numbers:

$$A = \ln(2^{-1074}) = -744.44007\ldots$$
$$B = \ln\left((1 - 2^{-53}).2^{1024}\right) = 709.78271\ldots$$

The exponential of a number larger than B is an overflow, whereas the exponential of a number less than A is rounded to 0.

However, subtler under/overflow situations may arise in two cases:

- An intermediate computation may raise an overflow although the final result is representable as an IEEE-754 floating-point number.
- In IEEE-754 arithmetic, when a result is between 2^{-1023} and 2^{-1074}, a gradual underflow exception arises to signal that the precision of the result is reduced in a drastic way.

In both cases, as will be shown in the following, it is possible to avoid the exception by predicting that it will occur, and appropriately scaling the input number in the range reduction phase.

3.3 First reduction step

The purpose of this first reduction step is to replace $x \in [A, B]$ with four floating-point numbers x_1, ℓ_0^ℓ, $\ell_0^{\ell\ell}$ and y_0 such that we have

- to be used in the **Quick** phase of the algorithm:

$$e^x = e^{x_1} \times y_0 \times e^{-\ell_0^\ell}(1+\delta_1) \qquad \text{with} \quad |\delta_1| \le 2^{-97}, \tag{1}$$

- to be used in the **Accurate** phase if needed:

$$e^x = e^{x_1} \times y_0 \times e^{-\ell_0^\ell - \ell_0^{\ell\ell}}(1+\epsilon_1) \qquad \text{with} \quad |\epsilon_1| \le 2^{-120}. \tag{2}$$

If $|x|$ is less than $3.2^1 = 6$, we may define $y_0 = 1$, $\ell_0^\ell = \ell_0^{\ell\ell} = 0$ and $x_1 = x$, and the relations (1) and (2) hold.

Otherwise, we compute the multiple $\hat{x}$ of 4 nearest to x (this can be done quickly). From $A < x < B$ we deduce that there are 363 possible different values for $\hat{x}$. We then read in a table $y_0 = 1 \times 2^{e_0}$ the floating-point number with a mantissa of 1 that is nearest to $e^{\hat{x}}$, and $\ell_0 = \ln(y_0)$. Since ℓ_0 is not exactly representable in floating-point arithmetic, we will represent it as the sum of two double precision numbers, ℓ_0^h and ℓ_0^ℓ, and one single precision number $\ell_0^{\ell\ell}$ defined as follows:

- ℓ_0^h is the double precision number closest to ℓ_0;
- ℓ_0^ℓ is the double precision number closest to $\ell_0 - \ell_0^h$;
- $\ell_0^{\ell\ell}$ is the single precision number closest to $(\ell_0 - \ell_0^h) - \ell_0^\ell$.

We have

$$x - 2 \le \hat{x} \le x + 2,$$

equivalent to

$$e^x e^{-2} \le e^{\hat{x}} \le e^x e^{+2},$$

hence

$$e^x e^{-2}\left(1 - 2^{-1}\right) \le y_0 \le e^x e^{+2}\left(1 + 2^{-1}\right),$$

which gives

$$x - 2 + \ln\left(1 - 2^{-1}\right) \le \ell_0 \le x + 2 + \ln\left(1 + 2^{-1}\right).$$

Now, from $-744.44\ldots \le x \le 709.78\ldots$, we deduce that $\mathrm{ulp}(x) \le 2^{-43}$. Therefore,

$$\begin{gathered} x - 2 - 0.694 - 2^{-44} \le \ell_0^h \le x + 2 + 0.405 + 2^{-44} \\ |\ell_0^\ell| \le 2^{-44} \\ |\ell_0^{\ell\ell}| \le 2^{-44-53} = 2^{-97} \\ |\ell_0^{\ell\ell} - (\ell_0^\ell - (\ell_0 - \ell_0^h))| \le 2^{-120} \end{gathered}$$

Besides, we notice that $x \geq 6$ implies $x/2 \leq x-2-0.694-2^{-44}$ and $x+2+0.405+2^{-44} \leq 2x$ which means $x/2 \leq \ell_0^h \leq 2x$. Therefore (using Sterbenz lemma), the number $x_1 = x \ominus \ell_0^h$ is exactly computed in double precision.

To sum it up, at the end of this step we have read the floating-point numbers y_0, ℓ_0^h, ℓ_0^ℓ and $\ell_0^{\ell\ell}$ and computed exactly $x_1 = x \ominus \ell_0^h$ such that the relations (1) and (2) hold, as well as the following relations:

$$\begin{cases} |x_1| \leq 3.2^1 = 6 \\ |\ell_0^\ell| \leq 2^{-44} \\ |\ell_0^{\ell\ell}| \leq 2^{-97} \\ y_0 \text{ is a floating-point number with a mantissa of 1} \end{cases}$$

3.4 Using the first step to avoid exceptions

Since $B < 710$, for $708 < x < B$ the multiple of 4 nearest to x will be $\hat{x} = 708$. Hence $y_0 = 2^{1021}$ will not cause an overflow in this step.

Concerning gradual underflow, we want to perform intermediate computations with at least 120 bits of precision. Therefore we know that we will have no gradual underflow problem if $x > C = \ln(2^{-1074+120}) = -661.2624\ldots$.

Before beginning the algorithm, we may test if x belongs to the interval $[A, C]$ on which gradual underflows may develop. In this case, we will have filled all the tables with values corresponding to $x+\ln(2^{120})$. This moves x from $[A, C]$ to $[C, B]$ in the rest of the algorithm. After computing the exponential of the new value and rounding correctly to double precision, we will just have to multiply the result by 2^{-120}. This is done exactly unless the result is a gradual underflow, in which case IEEE-754 arithmetic guarantees correct rounding nevertheless.

The three other reduction steps will be very similar to the first, and the computations will be less detailed.

3.5 Second reduction step

If $|x_1| \leq 3 \times 2^{-7}$ we define $y_1 = 1$, $\ell_1 = 0$ and $x_2 = x_1$.

Otherwise, we compute the multiple $\hat{x_1}$ of 2^{-6} that is closest to x_1, and we read in a table y_1 (the floating-point number with a 8-bit mantissa closest to $e^{\hat{x_1}}$) and $\ell_1 = \ln(y_1)$. ℓ_1 is actually stored as ℓ_1^h (the double precision number closest to ℓ_1), ℓ_1^ℓ (the double precision number closest to $\ell_1 - \ell_1^h$), and $\ell_1^{\ell\ell}$ (the single precision number closest to $(\ell_1 - \ell_1^h) - \ell_1^\ell$). In a way very similar to the previous section, we get

$$x_1 - 2^{-7} - 2^{-8} - 2^{-16} - 2^{-51} < \ell_1^h < x_1 + 2^{-7} + 2^{-8} + 2^{-51}$$

which ensures that the subtraction $x_2 = x_1 \ominus \ell_1^h$ is performed exactly, without any rounding error. Hence, after step 2, we have four floating-point numbers x_2, ℓ_1^ℓ, $\ell_1^{\ell\ell}$ and $Y_1 = y_0 \times y_1$ satisfying

$$\begin{cases} |x_2| < 3 \times 2^{-7} \\ |\ell_1^\ell| \leq 2^{-51} \\ |\ell_1^{\ell\ell}| \leq 2^{-104} \\ Y_1 \text{ is a floating-point number with a 9-bit mantissa} \end{cases}$$

such that we have (to be used in the **Quick** phase of the algorithm)

$$e^x = e^{x_2} \times Y_1 \times e^{-\ell_0^\ell - \ell_1^\ell}(1+\delta_1)(1+\delta_2) \qquad \text{with} \quad |\delta_2| \leq 2^{-104},$$

and (to be used in the **Accurate** phase if needed):

$$e^x = e^{x_2} \times Y_1 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_0^{\ell\ell} - \ell_1^{\ell\ell}}(1+\epsilon_1)(1+\epsilon_2) \qquad \text{with} \quad |\epsilon_2| \leq 2^{-127}.$$

3.6 Third reduction step

If $|x_2| \leq 3 \times 2^{-16}$ then we define $y_2 = 1$, $\ell_2 = 0$ and $x_3 = x_2$

Otherwise, we compute the multiple $\hat{x_2}$ of 2^{-15} that is closest to x_2, and read y_2 (the floating-point number with a 17-bit mantissa number closest to $e^{\hat{x_2}}$) and $\ell_2 = \ln(y_2)$. We also define ℓ_2^h as the double precision number closest to ℓ_2 and ℓ_2^ℓ as the double precision number closest to $\ell_2 - \ell_2^h$. As for the previous steps, we find that the subtraction $x_3 = x_2 \ominus \ell_2^h$ is performed exactly, and we finally get three variables x_3, ℓ_2^ℓ and $Y_2 = y_0 \times y_1 \times y_2$ satisfying

$$\begin{cases} |x_3| < 3 \times 2^{-16} \\ |\ell_2^\ell| \leq 2^{-59} \\ Y_2 \text{ is a floating-point number with a 26-bit mantissa} \end{cases}$$

such that we have, to be used respectively in the **Quick** and **Accurate** phases :

$$\begin{cases} e^x = e^{x_3} \times Y_2 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell}(1+\delta_1)(1+\delta_2)(1+\delta_3) & \text{with} \quad |\delta_3| \leq 2^{-112} \\ e^x = e^{x_3} \times Y_2 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_0^{\ell\ell} - \ell_1^{\ell\ell}}(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) & \text{with} \quad |\epsilon_3| \leq 2^{-112} \end{cases}$$

3.7 Fourth reduction step

If $|x_3| \leq 3 \times 2^{-26}$, we define $y_3 = 1$, $\ell_3 = 0$ and $t = x_3$.

Otherwise, we compute the multiple $\hat{x_3}$ of 2^{-25} that is closest to x_3, and read y_3 as the floating-point number with a 27-bit mantissa closest to $e^{\hat{x_3}}$ and ℓ_3 as $\ln(y_3)$. We also define ℓ_3^h as the double precision number closest to ℓ_3 and ℓ_3^ℓ as the double precision number closest

to $\ell_3 - \ell_3^h$. As for the previous steps, we find that the subtraction $t = x_3 \ominus \ell_3^h$ is performed exactly, and we finally get three variables t, ℓ_3^ℓ and $Y_3 = y_0 \times y_1 \times y_2 \times y_3$ satisfying

$$\begin{cases} |t| < 3 \times 2^{-26} \\ |\ell_3^\ell| \leq 2^{-69} \\ Y_3 \text{ is a floating-point number with a 53-bit mantissa} \end{cases}$$

such that we have, to be used respectively in the **Quick** and **Accurate** phases :

$$\begin{cases} e^x = e^t \times Y_3 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell}(1+\delta_1)(1+\delta_2)(1+\delta_3)(1+\delta_4) & \text{with} \quad |\delta_4| \leq 2^{-122} \\ e^x = e^t \times Y_3 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell - \ell_0^{\ell\ell} - \ell_1^{\ell\ell}}(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) & \text{with} \quad |\epsilon_4| \leq 2^{-122} \end{cases}$$

3.8 Summing it up

Finally, in the **Quick** phase, we will use :

$$e^x = e^t \times Y_3 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell}(1+\delta_5) \tag{3}$$

with $|\delta_5| \leq |\delta_1| + |\delta_2| + |\delta_3| + |\delta_4| = 2^{-97} + 2^{-104} + 2^{-112} + 2^{-122}$.

In the **Accurate** phase of the algorithm, we will use :

$$e^x = e^t \times Y_3 \times e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell - \ell_0^{\ell\ell} - \ell_1^{\ell\ell}}(1+\epsilon_5) \tag{4}$$

with $|\epsilon_5| \leq |\epsilon_3| + |\epsilon_1| + |\epsilon_4| + |\epsilon_2| = 2^{-112} + 2^{-120} + 2^{-122} + 2^{-126}$.

The cost of the range reduction is that of computing the $\hat{x}_i$'s, reading the various values in tables, plus one subtraction in each step to perform the range reduction itself.

Most table values are used in both phases. The first step for instance needs a $(744 + 709)/4$=363 entry table, with for each entry two double precision numbers, and two single precision numbers. This corresponds to an 8 KBytes table. Similarly the second table is also composed with two double precision numbers, and two single precision numbers, whereas the third table is composed with two double precision numbers and one single precision number. Each entry in the fourth table is 3 double precision numbers. The sizes of these tables are summarized below:

Table	Nb of elem.	Size of each entry (in octet)	Total (in bytes)
First	363	24	8712
Second	768	24	18432
Third	1536	20	30720
Fourth	3072	24	73728
Total			131 Kb

These sizes can be reduced. Firstly, the y_i's contain many zeroes which need not be stored. Then storing $1-\ell_1$, $1-\ell_2$, $1-\ell_3$, would lead to a 25 bit reduction for the fourth table, 15 for the third table, and 6 for the second table. Finally, we may accept that $\epsilon_1 = \epsilon_4 = \epsilon_2 = 2^{-114}$ to align all the error bounds on the worst one. Together these ideas reduce the table to less than 100 KB, at the expense of additional operations. This remains to be evaluated.

4 Polynomial evaluation

After the reduction, we have obtained a number t whose absolute value is less than 3×2^{-26}. In this section we show how to approximate e^t in $\left[-3 \times 2^{-26}, +3 \times 2^{-26}\right]$ by a polynomial, of degree 2 in the **Quick** phase, and of degree 4 in the **Accurate** phase.

4.1 Quick phase

For the **Quick** phase we use this second degree polynomial :

$$P_1(t) = 1 + t + \frac{1}{2}t^2, \tag{5}$$

with the approximation error :

$$\max_{t \in [-3\times 2^{-26}, +3\times 2^{-26}]} \left|P(t) - e^t\right| \quad = \quad 1.49 \times 10^{-22} \quad < \quad 2^{-75} \tag{6}$$

We evaluate this polynomial by the following expression, where $\oplus$, and $\otimes$ denote the conventional arithmetic operators, correctly rounded to nearest as specified by the IEEE-754 standard:

$$R_1 = t \oplus (\frac{1}{2} \otimes (t \otimes t))$$

Theorem 4.1 (Polynomial evaluation error)
$1 + R_1$ approximates $1 + t + \frac{1}{2}t^2$ with an error less than 2^{-77}.

From this result and (6) we get:

Theorem 4.2 (Approximation error)

$$\textit{For} \;\; t \in [-3 \times 2^{-26}, +3 \times 2^{-26}], \qquad 1 + R_1 = e^t\,(1 + \delta_6) \quad \textit{with} \quad |\delta_6| \leq 2^{-74} \tag{7}$$

Proof. [4.2]
We have $|t| < 3 \times 2^{-26}$, hence $|t^2| < 9 \times 2^{-52} < 2^{-48}$, we deduce $\mathrm{ulp}(t^2) \leq 2^{-101}$. This gives,

$$\left|S - t^2\right| \leq 2^{-102}$$

Now dividing S by 2 to get K_1 is done exactly, then,

$$\left| K_1 - \frac{t^2}{2} \right| \leq 2^{-103} \tag{8}$$

From $|t| < 3 \times 2^{-26}$ and $|K_1| < 9 \times 2^{-53}$, we deduce $|t + K_1| < 2^{-23}$. Therefore,

$$|R_1 - (t + K_1)| \leq 2^{-77}$$

By combining this last result with (8) we get,

$$\left| R_1 - (t + \frac{t^2}{2}) \right| \leq 2^{-77} + 2^{-103}$$

So, the error induced by the computation of R_1 is less than $2^{-77} + 2^{-103}$. Combine with the approximation error which is less than 2^{-75}, we deduce that the error δ_6 commited by evaluating e^t, for $t \in [-3 \times 2^{-26},\ 3 \times 2^{-26}]$ by $1 + R_1$, is :

$$\delta_6 < 2^{-74}$$

□

4.2 Accurate phase

Here more accuracy is needed, and we therefore approximate e^t by a constrained minimax polynomial [13]:

$$P(t) = 1 + t + \frac{1}{2}t^2 + a_3 t^3 + a_4 t^4, \tag{9}$$

where $a_3 = \frac{12009599006321323}{72057594037927936}$ and $a_4 = \frac{12009599006321323}{288230376151711744}$ computed in Maple are exactly representable in double precision.

The approximation error is now

$$\max_{t \in [-3\times 2^{-26}, +3\times 2^{-26}]} \left| P(t) - e^t \right| \quad = \quad 1.07 \times 10^{-38} \quad < \quad 2^{-129} \tag{10}$$

The following algorithm evaluates this polynomial using only $\oplus$ and $\otimes$. All the operations are assumed correctly rounded to nearest, and some are exact (division by 2 and Fast_two_sum). Also remark that $t \otimes t$ has already been computed in the **Quick** phase.

$$\begin{aligned}
K_1 &= a_4 \otimes t \\
K_2 &= a_3 \oplus K_1 \\
S &= t \otimes t \\
K_3 &= K_2 \otimes S \\
R_1 &= t \otimes K_3 \\
(S^h,\ S^\ell) &= \text{Dekker}(t,\ t) \\
Q^h &= S^h/2 \\
Q^\ell &= S^\ell/2 \\
R_2 &= R_1 \oplus Q^\ell \\
(R_3,\ R_4) &= \text{Fast_two_sum}(R_2,\ Q^h) \\
(P_1,\ A) &= \text{Fast_two_sum}(R_3,\ t) \\
P_2 &= A \oplus R_4
\end{aligned}$$

This sequence involves 11 $\otimes$, 21 $\oplus$ and 2 shifts. Its precision is given by the following theorems:

Theorem 4.3 (Polynomial evaluation error)
$1 + P_1 + P_2$ approximates $1 + t + \frac{1}{2}t^2 + a_3t^3 + a_4t^4$ with an error less than 2^{-126}.

From this result and (10) we get:

Theorem 4.4 (Approximation error)

$$\textit{For} \quad t \in [-3 \times 2^{-26}, +3 \times 2^{-26}], \quad 1 + P_1 + P_2 = e^t\,(1 + \epsilon_6) \qquad \textit{with} \quad |\epsilon_6| \leq 2^{-125}. \quad (11)$$

Proof. [4.4]
We have the folowing property, where Δx is the error induced by the approximation of x in floating-point arithmetic.

$$\begin{aligned}
P_1 + P_2 &= P_1 + A + R_4 + \Delta P_2 \\
&= t + R_3 + R_4 + \Delta P_2 \\
&= t + R_2 + Q^h + \Delta P_2 \\
&= t + R_1 + Q^h + Q^\ell + \Delta R_2 + \Delta P_2 \\
&= \tfrac{S^h}{2} + \tfrac{S^\ell}{2} + t + tK_3 + \Delta R_1 + \Delta R_2 + \Delta P_2 \\
&= \tfrac{t^2}{2} + t + t(K_2S + \Delta K_3) + \Delta R_1 + \Delta R_2 + \Delta P_2 \\
&= \tfrac{t^2}{2} + t + t((a_3 + K_1 + \Delta K_2)(t^2 + \Delta S) + \Delta K_3) + \Delta R_1 + \Delta R_2 + \Delta P_2 \\
&= \tfrac{t^2}{2} + t + t((a_3 + (a_4t + \Delta K_1) + \Delta K_2)(t^2 + \Delta S) + \Delta K_3) + \Delta R_1 + \Delta R_2 + \Delta P_2 \\
&= t + \tfrac{t^2}{2} + a_3t^3 + a_4t^4 + \Delta Res.
\end{aligned}$$

With

$$\begin{aligned}
\Delta Res &= t^3\Delta K_1 + t^3\Delta K_2 + t\Delta K_3 + t\Delta S(a_3 + a_4t + \Delta K_1 + \Delta K_2) + \\
&\quad \Delta R_1 + \Delta R_2 + \Delta P_2
\end{aligned}$$

Now let us compute the error due to the computation.
The maximum possible value of $|a_4 t|$ is around $1.86 \times 10^{-8} \approx 2^{-25}$. Hence $|a_4 t| < 2^{-24}$. Therefore, $\mathrm{ulp}(a_4 t) \leq 2^{-77}$ and

$$\Delta K_1 = |K_1 - a_4 t| \leq 2^{-78} \tag{12}$$

Now from, $|a_3 + K_1| < 1/4$ we have $\mathrm{ulp}(a_3 + K_1) < 2^{-55}$, then

$$\Delta K_2 = |K_2 - (a_3 + K_1)| \leq 2^{-56} \tag{13}$$

Combining (12) and (13) we have :

$$|K_2 - (a_3 + a_4 t)| \leq 2^{-56} + 2^{-78}$$

From $|t^2| < 9 \times 2^{-52} < 2^{-48}$, we deduce $\mathrm{ulp}(t^2) \leq 2^{-101}$. Hence,

$$\Delta S = |S - t^2| \leq 2^{-102}$$

Now from

$$|S \times K_2| < \left(9 \times 2^{-52} + 2^{-102}\right) \times \left(0.1666666666666666671 \ldots + 2^{-56} + 2^{-78}\right) < 2^{-51}$$

We deduce

$$\Delta K_3 = |K_3 - S K_2| \leq 2^{-105}$$

Now, from $|K_3| < 2^{-51}$ and $|t| \leq 3 \times 2^{-26}$, we deduce $|K_3 t| < 2^{-75}$. Therefore,

$$\Delta R_1 = |R_2 - K_3 t| \leq 2^{-129} \tag{14}$$

Now from $|R_1| < 2^{-75}$ and $|Q^\ell| < 2^{-101}$ we deduce $|R_1 + Q^\ell| < 2^{-74}$. Hence ,

$$\Delta R_2 = \left|R_2 - (R_1 + Q^\ell)\right| \leq 2^{-128}$$

The values R_3 and R_4 come from the *Fast_two_sum* algorithm for computing the exact sum of the two floating-point number R_2 and Q^h. Hence $R_3 \oplus R_4 = R_2 + Q^h$ exactly, with $|R_3| < 2^{-48}$, and $|R_4| < 2^{-101}$.

In a similar way, $P_1 \oplus A = R_3 + t$ exactly. Moreover, from $|R_3| < 2^{-48}$ and $|t| < 3 \times 2^{-26}$, we have $|P_1| < 2^{-23}$ and $|A| < 2^{-76}$.
Finally, from $|A| < 2^{-76}$ and $|R_4| < 2^{-101}$ we have $|R_4 + A| < 2^{-75}$. then

$$\Delta P_2 = |P_2 - (A + R_4)| \leq 2^{-129}$$

Therefore the overall error is less than :

$$\begin{aligned}
\Delta Res &= t^3 \Delta K_1 + t^3 \Delta K_2 + t \Delta K_3 + t \Delta S (a_3 + a_4 t + \Delta K_1 + \Delta K_2) + \\
&\quad \Delta R_1 + \Delta R_2 + \Delta P_2 \\
&\leq 2^{-73} 2^{-78} + 2^{-73} 2^{-56} + 2^{-24} 2^{-105} + 2^{-24} 2^{-102} (a_3 + a_4 2^{-24} + 2^{-78} + 2^{-56}) + \quad pp \\
&\quad 2^{-129} + 2^{-128} + 2^{-129} \\
&\leq 2^{-126}
\end{aligned}$$

□

5 Evaluation of the remaining terms

At this point, we remind the reader that we wish to evaluate e^x using (3) and (4). The two previous section have shown how to compute respectively Y_3 and e^t (for both the **Quick** and **Accurate** phases). We now have to evaluate all the correcting terms which comes from the table reduction step. This terms are ℓ_i^ℓ or $\ell_j^{\ell\ell}$ for $i \in \{0,\ 1,\ 2,\ 3\}$ or $j \in \{0,\ 1\}$. Here again there will be a **Quick** evaluation and an **Accurate** one.

5.1 Quick evaluation of $e^{-\ell_0^\ell-\ell_1^\ell-\ell_2^\ell-\ell_3^\ell}$

The reduction steps have provided the following bounds:

$$|\ell_0^\ell| \le 2^{-44} \qquad |\ell_1^\ell| \le 2^{-51} \qquad |\ell_2^\ell| \le 2^{-59} \qquad |\ell_3^\ell| \le 2^{-69}$$

This leads us to the following order of operations to compute the sum (always rounded to nearest):

- $L_0 = \ell_3^\ell \oplus \ell_2^\ell$. Since $\mathrm{ulp}(\ell_3^\ell + \ell_2^\ell) \le 2^{-111}$, the error incurred by this computation is $\left|L_0 - (\ell_3^\ell + \ell_2^\ell)\right| \le 2^{-112}$.
- $L_1 = \ell_1^\ell \oplus L_0$. Since $\mathrm{ulp}(\ell_1^\ell + L_0) \le 2^{-103}$, the error is $\left|L_1 - (L_0 + \ell_1^\ell)\right| \le 2^{-104}$.
- $L_2 = \ell_0^\ell \oplus L_1$. Since $\mathrm{ulp}(\ell_0^\ell + L_1) \le 2^{-96}$, the error is $\left|L_2 - (L_1 + \ell_0^\ell)\right| \le 2^{-97}$.

Combining all three bounds, we get

$$\left|L_2 - \left(\ell_0^\ell + \ell_1^\ell + \ell_2^\ell + \ell_3^\ell\right)\right| \le 2^{-97} + 2^{-103}. \tag{15}$$

Now we want to evaluate e^{-L_2}. From (15) we deduce,

$$\left|e^{-L_2} - e^{-\ell_0^\ell-\ell_1^\ell-\ell_2^\ell-\ell_3^\ell}\right| \le \left(2^{-97} + 2^{-103}\right) \times e^\xi,$$

where $\xi \in \left[-L_2, -\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell\right]$. Since $|\xi| \le 2^{-44} + 2^{-51} + 2^{-59} + 2^{-69} + 2^{-97} + 2^{-103}$, we have $e^\xi \le 1 + 2^{-44} + 2^{-50}$. Therefore

$$\left|e^{-L_2} - e^{-\ell_0^\ell-\ell_1^\ell-\ell_2^\ell-\ell_3^\ell}\right| \le 2^{-97} + 2^{-102}. \tag{16}$$

Now we approximate $e^{-L_2} - 1$ using the same degree-2 polynomial as used for e^t. Then we compute:

- $W_1 = L_2 \otimes L_2$
- $W_2 = W_1/2$
- $W_3 = W_2 \ominus L_2$

Its precision is given by the following theorems:

Theorem 5.1 (Approximation error)
$1 + W_3$ *approximates* $e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell}$ *with an error less than* $2^{-96} + 2^{-104}$. *This can be rewritten as*

$$1 + W_3 = e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell}(1 + \delta_7) \qquad \textit{with} \quad |\delta_7| \leq 2^{-96} + 2^{-104}. \tag{17}$$

Proof. [5.1]
We have: $(L_2)^2 \leq 2^{-88} + 2^{-93}$, therefore $\mathrm{ulp}((L_2)^2) \leq 2^{-140}$. This gives,

$$\left| W_2 - \frac{(L_2)^2}{2} \right| \leq 2^{-142}$$

and, since $\mathrm{ulp}(L_2 + W_2) \leq 2^{-96}$, we have

$$|W_3 - (-L_2 + W_2)| \leq 2^{-97}.$$

We therefore get

$$\left| W_3 - \left(-L_2 + \frac{(L_2)^2}{2} \right) \right| \leq 2^{-97} + 2^{-142}. \tag{18}$$

To obtain the desired result, it now suffices to evaluate the difference between $e^{-L_2} - 1$ and $(-L_2 + (L_2)^2/2)$. This difference is equal to

$$\frac{(L_2)^3}{6} e^{\zeta},$$

where $\zeta \in [0, L_2]$. This gives

$$\left| e^{-L_2} - \left(1 - L_2 + \frac{(L_2)^2}{2} \right) \right| \leq 2^{-133}. \tag{19}$$

Combining (18) and (19), we get

$$\left| e^{-L_2} - (1 + W_3) \right| \leq 2^{-97} + 2^{-132}.$$

Therefore, using (16),

$$\left| e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell} - (1 + W_3) \right| \leq 2^{-96} + 2^{-101}.$$

This can be rewritten as

$$1 + W_3 = e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell - \ell_3^\ell}(1 + \delta_7) \qquad \text{with} \quad |\delta_7| \leq 2^{-96} + 2^{-104}.$$

□

5.2 Accurate evaluation of $e^{-\ell_0^\ell-\ell_1^\ell-\ell_2^\ell-\ell_3^\ell-\ell_0^{\ell\ell}-\ell_1^{\ell\ell}}$

We now have to evaluate e^x by a method more accurate than the previous one using (4). We already know Y_2 and e^t. We now have to evaluate $e^{(-\ell_0^\ell-\ell_1^\ell-\ell_2^\ell-\ell_3^\ell-\ell_0^{\ell\ell}-\ell_1^{\ell\ell})}$.

5.2.1 Computation of $\ell_0^\ell+\ell_1^\ell+\ell_2^\ell+\ell_3^\ell+\ell_0^{\ell\ell}+\ell_1^{\ell\ell}$

We firstly compute $\ell_0^\ell+\ell_1^\ell+\ell_2^\ell+\ell_3^\ell+\ell_0^{\ell\ell}+\ell_1^{\ell\ell}$ on a pseudo-expansion on lenght two. We have:

$$\begin{cases} |\ell_0^\ell| & \leq & 2^{-44} \\ |\ell_1^\ell| & \leq & 2^{-51} \\ |\ell_2^\ell| & \leq & 2^{-59} \\ |\ell_3^\ell| & \leq & 2^{-69} \\ |\ell_0^{\ell\ell}| & \leq & 2^{-97} \\ |\ell_1^{\ell\ell}| & \leq & 2^{-104} \end{cases}$$

We will successively compute:

$$\begin{array}{rcl} L_0 & = & \ell_0^{\ell\ell} \oplus \ell_1^{\ell\ell} \\ L_1 & = & \ell_3^\ell \oplus L_0 \\ (L_2^h,\ L_2^\ell) & = & \text{Fast\_two\_sum}(\ell_2^\ell,\ L_1) \\ (A,\ B) & = & \text{Fast\_two\_sum}(\ell_1^\ell,\ L_2^h) \\ (L_4^h,\ C) & = & \text{Fast\_two\_sum}(A,\ \ell_0^\ell) \\ D & = & L_2^\ell \oplus B \\ L_4^\ell & = & D \oplus C \end{array}$$

Since $\text{ulp}(\ell_0^{\ell\ell}+\ell_1^{\ell\ell}) \leq 2^{-148}$, we deduce

$$\left|L_0 - (\ell_0^{\ell\ell}+\ell_1^{\ell\ell})\right| \leq 2^{-149}.$$

Since $\text{ulp}(\ell_3^\ell + L_0) \leq 2^{-120}$, we deduce

$$\left|L_1 - (\ell_3^\ell + L_0)\right| \leq 2^{-121}.$$

From the property of *Fast_two_sum* algorithm, we have

$$\begin{array}{rcl} L_2^h \oplus L_2^\ell & = & \ell_2^\ell + L_1 \\ A \oplus B & = & \ell_1^\ell + L_2 \\ L_4^h \oplus C & = & A + \ell_0^\ell \end{array}$$

with

- $|L_2^h| \leq 2^{-59} + 2^{-68}$ and $|L_2^\ell| \leq 2^{-110}$
- $|A| \leq 2^{-51} + 2^{-58}$ and $|B| \leq 2^{-102}$,

- $|L_4^h| \leq 2^{-44} + 2^{-50}$ and $|C| \leq 2^{-95}$.

We can deduce from that bound $|L_2^\ell + B| \leq 2^{-102} + 2^{-110}$, then

$$\left|D - (L_2^\ell + B)\right| \leq 2^{-154}$$

thus $|D + C| \leq 2^{-95} + 2^{-101}$, then

$$\left|L_4^\ell - (D + C)\right| \leq 2^{-148}$$

Combining all the previous result, we get,

$$\left|(L_4^h + L_4^\ell - \left(\ell_0^{\ell\ell} + \ell_1^{\ell\ell} + \ell_3^\ell + \ell_2^\ell + \ell_1^\ell + \ell_0^\ell\right)\right| \leq 2^{-121} + 2^{-147}. \tag{20}$$

5.2.2 Evaluation of $e^{-L_4^h - L_4^\ell}$

From (20) we deduce,

$$\left|e^{-L_4^h - L_4^\ell} - e^{\ell_0^{\ell\ell} + \ell_1^{\ell\ell} + \ell_3^\ell + \ell_2^\ell + \ell_1^\ell + \ell_0^\ell}\right| \leq \left(2^{-121} + 2^{-147}\right) \times e^\xi,$$

where $\xi \in \left[-L_4^h - L_4^\ell, -\ell_0^{\ell\ell} - \ell_1^{\ell\ell} - \ell_3^\ell - \ell_2^\ell - \ell_1^\ell - \ell_0^\ell\right]$. Since $|\xi| \leq 2^{-44} + 2^{-50}$, we have $e^\xi \leq 1 + 2^{-44} + 2^{-50}$. Therefore

$$\left|e^{-L_4^h - L_4^\ell} - e^{\ell_0^{\ell\ell} + \ell_1^{\ell\ell} + \ell_3^\ell + \ell_2^\ell + \ell_1^\ell + \ell_0^\ell}\right| \leq 2^{-121} + 2^{-146}. \tag{21}$$

Now, let us approximate $e^{-L_4^h - L_4^\ell} - 1$ using the same degree-2 polynomial approximation as used for e^t. Then we compute:

$$\begin{array}{rcl} W_1 & = & L_4^h \otimes L_4^h \\ W_2 & = & W_1/2 \\ W_4^h & = & -L_4^h \\ W_4^\ell & = & W_2 \ominus L_4^\ell \end{array}$$

From $|L_4^h| \leq 2^{-44} + 2^{-50}$ we deduce $|(L_4^h)^2| \leq 2^{-87}$, $\mathrm{ulp}(W_1) \leq 2^{-139}$ and

$$|W_1 - (L_4^h)^2| \leq 2^{-140} \tag{22}$$

Since a division by 2 is done exactly we have $|W_2 - \frac{(L_4^h)^2}{2}| \leq 2^{-141}$.
From $|L_4^\ell| \leq 2^{-95} + 2^{-101}$ and $|W_2| \leq 2^{-88}$ we deduce that

$$\left|W_4^\ell - (W_2 - L_4^\ell)\right| \leq 2^{-140} \tag{23}$$

Combining (22) and (23) we get :

$$\left|(W_4^h + W_4^\ell) - (\frac{(L_4^h)^2}{2} - L_4^\ell - L_4^h)\right| \leq 2^{-139} \tag{24}$$

Since $|L_4^h| \leq 2^{-44} + 2^{-50}$ and $|L_4^\ell| \leq 2^{-95} + 2^{-101}$ we have :

- $|(L_4^\ell)^2| \leq 2^{-189}$
- $|L_4^h \times L_4^\ell| \leq 2^{-138}$

From (24) and previous bound we get :

$$\left|(W_4^h + W_4^\ell) - (\frac{(L_4^h + W_4^\ell)^2}{2} - L_4^\ell - L_4^h)\right| \leq 2^{-137} \tag{25}$$

To obtain the desired result, it now suffices to evaluate the difference between $e^{-L_4^h - L_4^\ell} - 1$ and $(\frac{(L_4^h + L_4^\ell)^2}{2} - L_4^\ell - L_4^h)$. This difference is equal to

$$\frac{(L_4^h + L_4^\ell)^3}{6} e^\zeta,$$

where $\zeta \in [0, -(L_4^h + L_4^\ell)]$. This gives

$$\left|e^{-(L_4^h + L_4^\ell)} - \left(1 - \frac{(L_4^h)^2}{2} - L_4^\ell - L_4^h\right)\right| \leq 2^{-133}. \tag{26}$$

Combining (25) and (26), we get

$$\left|e^{-(L_4^h + L_4^\ell)} - \left(1 + W_4^h + W_4^\ell\right)\right| \leq 2^{-132}.$$

Therefore, using (21),

$$\left|e^{-\ell_0^\ell - \ell_1^\ell - \ell_2^\ell} - \left(1 + W_4^h + W_4^\ell\right)\right| \leq 2^{-121} + 2^{-131}.$$

This can be rewritten as

$$1 + W_4^h + W_4^\ell = e^{\ell_0^{\ell\ell} + \ell_1^{\ell\ell} + \ell_3^\ell + \ell_2^\ell + \ell_1^\ell + \ell_0^\ell}(1 + \epsilon_7) \qquad \text{with} \quad |\epsilon_7| \leq 2^{-121} + 2^{-131} \tag{27}$$

6 Reconstruction

Now we have all the elements for computing the exponential on double precision. As for previous sections, we divided that one in a **Quick** phase and an **Accurate** one.

6.1 Quick phase

In the previous sections, we have computed double-precision numbers Y_3, R_1, and W_3 such that

$$e^x = Y_3 \times (1 + R_1) \times (1 + W_3)\,(1 + \delta_8) \tag{28}$$

where $(1+\delta_8) = (1+\delta_5)(1+\delta_6)(1+\delta_7) \leq (1+2^{-97}+2^{-103})(1+2^{-75}+2^{-77}+2^{-103})(1+2^{-96}+2^{-104})$, which gives $|\delta_8| < 2^{-74}$. We also have the following bounds:

$$|R_1| < 2^{-23} \qquad \text{and} \qquad |W_3| < 2^{-43}$$

We approximate (28) by performing the following computations, where $\oplus$, $\ominus$ and $\otimes$ are rounded to nearest:

$$\begin{aligned} P_1 &= R_1 \otimes W_3 \\ P_2 &= W_3 \oplus P_1 \\ P_3 &= R_1 \oplus P_2 \\ (P_4^h,\ P_4^\ell) &= \text{Dekker}(Y_3,\ P_3) \\ (P_5^h, A) &= \text{fast\_two\_sum}(Y_3,\ P_4^h) \\ P_5^\ell &= A \oplus P_4^\ell \end{aligned}$$

Theorem 6.1 (Approximation error)
$P_5^h + P_5^\ell$ approximates e^x, for $x \in [-744.44\ldots,\ 709.78\ldots]$, with an error less than 2^{-73}.

Proof. [6.1]
From $|R_1| < 2^{-23}$ and $|W_3| < 2^{-43}$ we have that $|R_1 \times W_3| < 2^{-66}$, and then

$$|P_1 - (R_1 \times W_3)| \leq 2^{-119}$$

And combine with $|W_3| < 2^{-43}$ we get $|W_3 + P_1| < 2^{-43} + 2^{-66}$ and

$$|P_2 - (W_3 + P_1)| \leq 2^{-96}$$

or,

$$|P_2 - (W_3 + (R_1 \times W_3))| \leq 2^{-96} + 2^{-119}$$

From $|R_1| < 2^{-23}$ and $|P_2| < 2^{-43} + 2^{-66}$ we deduce $|R_1 + P_2| < 2^{-23} + 2^{-43}$

$$|P_3 - (R_1 + P_2)| < 2^{-76}$$

From *Dekker's* multiplication algorithm property we have $P_4^h \oplus P_4^\ell = Y_3 \times P_3$ exactly. Moreover we know that $|P_3| < 2^{-22}$, however since we're computing the exponential on the whole range of double precision floating-point number, we just know that $2^{-1074} < |Y_3| < 2^{1024}$. In the other cases an exception appears.
Let y be the exponent of Y_3 with $y \in [-1074,\ 1024]$ in the case where Y_3 is not an exception. In the same way we added Y_3, P_4^h by the fast_two_sum algorithm which allows us to have $P_5^h \oplus A = Y_3 + P_4^h$ exactly.
Since $|P_3| < 2^{-22}$, we have $|Y_3| > |Y_3 \times P_3| = |P_4^h + P_4^\ell|$, then we can deduce that :

- $|P_4^h| < 2^{y-22}$

- $|P_4^\ell| < 2^{y-75}$
- $|P_5^h| < 2^{y+1}$
- $|A| < 2^{y+1-53}$

Combining previous result we get :

$$|P_5^\ell - (A + P_4^\ell)| \leq 2^{y-104}$$

or, substituing A and P_4^ℓ by there value :

$$\begin{array}{rcl}
|(P_5^h + P_5^\ell) - (P_5^h + A + P_4^\ell)| & \leq & 2^{y-104} \\
|(P_5^h + P_5^\ell) - (Y_3 + P_4^h + P_4^\ell)| & \leq & 2^{y-104} \\
|(P_5^h + P_5^\ell) - (Y_3 + Y_3 \times P_3)| & \leq & 2^{y-104} \\
|(P_5^h + P_5^\ell) - (Y_3 + Y_3(R_1 + P_2))| & \leq & 2^{y-76} + 2^{y-104} \\
|(P_5^h + P_5^\ell) - (Y_3 + Y_3(R_1 + W_3 + P_1))| & \leq & 2^{y-76} + 2^{y-96} + 2^{y-104} \\
|(P_5^h + P_5^\ell) - (Y_3 + Y_3(R_1 + W_3 + R_1 \times W_3))| & \leq & 2^{y-76} + 2^{y-96} + 2^{y-104} + 2^{y-119} \\
& \leq & 2^{y-75}
\end{array}$$

The error induced by the computation of the formula is 2^{y-75}. Combining it with (28) we have shown that at the end of all this first methods we have 73 correct bits of the result. □

6.2 Accurate phase

Now, let us show how to do. In the previous sections, we have computed double-precision numbers Y_3, P_1, P_2, W_4^h, W_4^ℓ such that

$$e^x = Y_3 \times (1 + P_1 + P_2) \times \left(1 + W_4^h + W_4^\ell\right)(1 + \epsilon_8) \tag{29}$$

where $(1+\epsilon_8) = (1+\epsilon_5)(1+\epsilon_6)(1+\epsilon_7) = (1+2^{-112}+2^{-119})(1+2^{-125})(1+2^{-121}+2^{-131})$, which gives $|\epsilon_8| < 2^{-111}$. We remind the reader the following bounds on the variables we manipulate:

- $|P_1| < 2^{-23}$;
- $|P_2| < 2^{-75}$;
- $|W_4^h| < 2^{-43}$;
- $|W_4^\ell| < 2^{-87}$.

We first approximate $(1 + P_1 + P_2) \times \left(1 + W_4^h + W_4^\ell\right) - 1$ by performing the following computations

$$\begin{array}{rcl} X_0 & = & P_1 \otimes W_4^\ell \\ X_1 & = & P_1 \otimes W_4^h \\ X_2 & = & P_2 \oplus W_4^\ell \\ X_3 & = & X_1 \oplus X_2 \\ (X_4^h,\ A) & = & \text{Fast\_two\_sum}(P_1,\ W_4^h) \\ B & = & A \oplus X_0 \\ X_4^\ell & = & B \oplus X_3 \end{array}$$

Theorem 6.2 (Approximation error)
$X_4^h + X_4^\ell$ approximates $(1 + P_1 + P_2) \times \left(1 + W_4^h + W_4^\ell\right) - 1$, with an error less than 2^{-117}.

Proof. [6.2]
From $|P_1| < 2^{-23}$ and $|W_4^\ell| < 2^{-87}$ we have $\left|P_1 \times W_4^\ell\right| < 2^{-110}$ hence,

$$\left|X_0 - (P_1 \times W_4^\ell)\right| \leq 2^{-163} \tag{30}$$

From $|P_1| < 2^{-23}$ and $|W_4^h| < 2^{-43}$ we have $\left|P_1 \times W_4^h\right| < 2^{-66}$ hence,

$$\left|X_1 - (P_1 \times W_4^h)\right| \leq 2^{-120} \tag{31}$$

From $|P_2| < 2^{-75}$ and $|W_4^\ell| < 2^{-87}$ we have $\left|P_2 + W_4^\ell\right| < 2^{-74}$, hence

$$\left|X_2 - (P_2 + W_4^\ell)\right| \leq 2^{-128} \tag{32}$$

Combining last both result (31) and (32) we deduce :

$$|X_3 - (X_1 + X_2)| \leq 2^{-119}$$

From the property of *Fast_two_sum* algorithm we have $X_4^h \oplus A = P_1 + W_4^h$ exactly with,

- $|X_4^h| < 2^{-22}$
- $|A| < 2^{-75}$

From $|A| < 2^{-75}$ and $|X_0| < 2^{-110}$ we get that $|A + X_0| < 2^{-74}$ and the error introduce :

$$|B - (A + X_0)| \leq 2^{-127}$$

From $|B| < 2^{-74}$ and $|X_3| < 2^{-65}$ we get that $|A + X_3| < 2^{-64}$

$$\left|X_4^\ell - (B + X_3)\right| \leq 2^{-118}$$

Then

$$\begin{aligned}
\left|(X_4^h + X_4^\ell) - (X_4^h + B + X_3)\right| &\leq 2^{-118} \\
\left|(X_4^h + X_4^\ell) - (X_4^h + A + X_0 + X_3)\right| &\leq 2^{-118} + 2^{-127} \\
\left|(X_4^h + X_4^\ell) - (P_1 + W_4^h + X_0 + +X_3)\right| &\leq 2^{-118} + 2^{-127} \\
\left|(X_4^h + X_4^\ell) - (P_1 + W_4^h + X_0 + X_1 + X_2)\right| &\leq 2^{-118} + 2^{-119} + 2^{-127} \\
\left|(X_4^h + X_4^\ell) - (P_1 + W_4^h + P_1 \times W_4^h + P_1 \times W_4^\ell + P_2 + W_4^\ell)\right| &\leq 2^{-118} + 2^{-119} + 2^{-120} \\
&\quad +2^{-127} + 2^{-128} + 2^{-163}
\end{aligned}$$

Since

- $|P_2 \times W_4^h| < 2^{-118}$
- $|P_2 \times W_4^\ell| < 2^{-162}$

We get

$$\left|(X_4^h + X_4^\ell) - ((1 + P_1 + P_2) \times \left(1 + W_4^h + W_4^\ell\right) - 1)\right| \leq 2^{-116}$$

□

Once we got the approximation of $\left((1 + P_1 + P_2) \times \left(1 + W_4^h + W_4^\ell\right) - 1\right)$ by $(X_4^h + X_4^\ell)$, we have to compute $Y_3 \times (X_4^h + X_4^\ell + 1)$. In order to approximate it we perform the following computations :

$$\begin{aligned}
(X_5^h,\ X_5^\ell) &= \text{Dekker}(Y_3,\ X_4^h) \\
X_6 &= Y_3 \otimes X_4^\ell \\
B &= X_5^\ell \oplus X_6 \\
(X_7^h,\ X_7^\ell) &= \text{Fast_two_sum}(X_5^h,\ B) \\
(X_8^h,\ X_8^\ell) &= \text{Fast_two_sum}(Y_3,\ X_7^h) \\
X_8^{\ell\ell} &= X_7^\ell
\end{aligned}$$

Theorem 6.3 (Approximation error)
$X_8^h + X_8^\ell + X_8^{\ell\ell}$ approximates e^x, for $x \in [-744.44\ldots,\ 709.78\ldots]$, with 110 bit of precision.

Proof. [6.3]
From the property of *Dekker* algorithms we have $X_5^h \oplus X_5^\ell = Y_3 \times X_4^h$ exactly. Moreover we know that $|X_4^h| < 2^{-22}$ and from the underflow step detection that $2^{-1024} < |Y_3| < 2^{1024}$. Then we have the following property :

$$|Y_3| > |Y_3 \times X_4^h|$$

It follow that

- $|X_5^h| \leq 2^{y-22}$,
- $|X_5^\ell| \leq 2^{y-74}$.

From $|X_4^\ell| < 2^{-65}$ we get that $|Y_3 \times X_4^\ell| \leq 2^{y-65}$, then

$$\left|X_6 - (Y_3 \times X_4^\ell)\right| \leq 2^{y-118}$$

From $|X_5^\ell| \leq 2^{y-74}$ and $|X_6| \leq 2^{y-65}$ we have

$$\left|B - (X_5^\ell + X_6)\right| \leq 2^{y-117}$$

From property of *Fast_two_sum* algorithm we deduce that $X_7^h \oplus X_7^\ell = X_5^h + B$ exactly. Moreover from $|X_5^h| \leq 2^{y-22}$ and $|B| \leq 2^{y-64}$ we deduce that :

- $|X_7^h| \leq 2^{y-21}$,
- $|X_7^\ell| \leq 2^{y-74}$.

For the same argument we have $X_8^h \oplus X_8^\ell = Y_3 + X_7^h$ is exact, with

- $|X_8^h| \leq 2^{y+1}$,
- $|X_8^\ell| \leq 2^{y-51}$.
- $|X_8^{\ell\ell}| \leq 2^{y-74}$.

Let show that $X_8^h + X_8^\ell + X_8^{\ell\ell}$ approximate e^x with an error less than 2^{-110}.From the way of computing X_8^h, X_8^ℓ, $X_8^{\ell\ell}$ we deduce that $(X_8^h + X_8^\ell + X_8^{\ell\ell}) = (Y_3 + X_5^h + X_5^\ell + X_6)$ exactly. Hence,

$$\begin{array}{rcl}
\left|(X_8^h + X_8^\ell + X_8^{\ell\ell}) - (Y_3 + X_5^h + X_5^\ell + X_6)\right| & \leq & 2^{y-116} \\
\left|(X_8^h + X_8^\ell + X_8^{\ell\ell}) - (Y_3 + Y_3 \times X_4^h + X_6)\right| & \leq & 2^{y-116} \\
\left|(X_8^h + X_8^\ell + X_8^{\ell\ell}) - (Y_3 + Y_3 \times X_4^h + Y_3 \times X_4^\ell)\right| & \leq & 2^{y-116} + 2^{y-117} \\
\left|(X_8^h + X_8^\ell + X_8^{\ell\ell}) - (Y_3 \times (1 + P_1 + P_2) \times (1 + W_4^h + W_4^\ell))\right| & \leq & 2^{-116} + 2^{y-116} + 2^{y-117}
\end{array}$$

Combined with (29) we get the final error :

$$|(X_8^h + X_8^\ell + X_8^{\ell\ell}) - e^x| \leq 2^{-110}$$

□

7 Operational cost, implementation and timings

7.1 Cost recapitulation

We summarize in the following table all the elementary operation involved in our algorithm.

	Quick phase					**Accurate** phase				
Part	+	×	shift	mem. access	mem. size	+	×	shift	mem. access	mem. size
Tabular reduction	4	0	0	12	126 KB	0	0	0	+2	+5 KB
Polynomial	1	1	1	0	0	+21	+11	+2	0	0
Remaining term	4	1	1	0	0	+17	+1	+1	0	0
Reconstruction	21	8	0	0	0	+31	+14	0	0	0
Total	30	10	2	12	126 KB	+69	+26	+3	+2	+5 KB

The actual cost of each operation is architecture-dependent. Besides the cost of memory accesses is very application-dependent, as it depends on the state of the cache.

There is probably room for improvement: Firstly, some of our steps waste precision, and this baseline algorithm could be further optimized. Secondly, depending on the architecture of the machine, the operation count could be greatly improved, either using fused multiply-and-add operations or hardware evaluation of the exponential, or exploiting double-extended precision to reduce the number of computation steps.

7.2 Implementation and timings

We made a preliminary implementation of this method, and we give below a comparison of this implementation with two similar libraries: `math.h`, the standard math library which does not provide correct rounding, and `MPFR`, a multiprecision library based on `GMP` that provides correct rounding but is not optimized specifically for double-precision. The following table shows execution times on a Sparc Ultra 5 at 400MHz with SunOS 5.8 and on an Intel Celeron at 566 Mhz under Linux, using gcc version 2.96. We normalized results to Celeron with `math.h` exponential computation.

	Sparc	Celeron
math.h	0.7	1
MPFR	106	59
our method	55	2.3

This table shows that the price to pay for correct rounding is a 2.3 times slower exponential on the Celeron machine, which is encouraging considering that typical applications involve few exponentials with respect to standard arithmetic operations. Until now we are unable to justify the difference of results for our method between Sparc and Celeron architectures. Besides, these preliminary test results are probably very dependent on compiler version, pipeline and cache considerations, and architecture in general. We hope to improve our implementation by taking into account these aspects.

7.3 Summary

Since we have given details of every part for computing the exponential, now we summarise every step of the algorithm:

Data : A double precision floating-point number $x = m.2^e$
Result: A double precision floating-point number *Res*
Exception detection begin
  if $e < A$ **then** Res = 1;
  if $A < e < C$ **then** Under=Yes;
  if $B < e$ **then** Overflow=Yes; STOP ;
end
$(t, Y_3, \ell_0^\ell, \ell_1^\ell, \ell_2^\ell, \ell_3^\ell)$ = **Table reduction**(x);
(R_1) = **Quick Polynomial evaluation**(t);
(W_3) = **Quick Remainding terms evaluation**$(\ell_0^\ell, \ell_1^\ell, \ell_2^\ell, \ell_3^\ell)$;
(P_5^h, P_5^ℓ) = **Quick Reconstruction**(Y_3, R_1, W_3);
if *Table Maker dilema don't occur for* $(P_5^h + P_5^\ell)$ **then**
  Res=Round(P_5^h, P_5^ℓ);
else
  if $x \in$ *Challenging cases* **then** *Res*=**Challenging cases**(x); STOP ;
  $(\ell_0^{\ell\ell}, \ell_1^{\ell\ell})$ = **Look up more terms**(x);
  (P_1, P_2) = **Accurate Polynomial evaluation**(t);
  (W_4^h, W_4^ℓ) = **Accurate Raimainding terms evaluation**$(\ell_0^\ell, \ell_1^\ell, \ell_2^\ell, \ell_3^\ell, \ell_0^{\ell\ell}, \ell_1^{\ell\ell})$;
  $(X_8^h, X_8^\ell, X_8^{\ell\ell})$ = **Accurate Reconstruction**(Y_3, R_1, W_3);
  Res=Round$(X_8^h, X_8^\ell, X_8^{\ell\ell})$;
end
if *Under=Yes* **then** $Res = Res/2^{-120}$;

Algorithm 1: Exponential computation

8 Conclusion

We have presented an algorithm and its implementation for the computation of exponentials in double precision with correct rounding to the nearest. It should be noted that the three other IEEE-754 rounding mode (round to $+\infty$, to $-\infty$, and to zero) are obtained by straightforward modifications in the last steps of this algorithm. The main goals were portability (based on IEEE-754 arithmetic), reasonable average speed when compared to the standard math libraries, and bounded worst-case time to allow using this algorithm in critical applications.

Initial tests show that the performance of our algorithm is intermediate between that of standard math libraries, which don't provide correct rounding, and MPFR, the only multiprecision package to date that provides correct rounding for the exponential.

One major drawback of our algorithm is the size of the table it needs, about 131KB, which is barely acceptable considering current cache sizes. We will be working on reducing this size in the near future.

It should also be noted that a faster implementation, using similar techniques, could be developped for processors implementing the IA-32 and IA-64 instruction sets, which provide "double extended" precision (64 bits of mantissa instead of 53). Similarly, the use of fused multiply-and-add operators, which can be found on an increasing number of architectures, could lead to an increase in precision, speed, and or reduction in table sizes.

Our current research directions, however, is rather to extend this portable approach to other transcendental functions (logarithm and trigonometric functions).

References

[1] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Library Reference Manual.* Springer Verlag, Berlin, Germany, 1991.

[2] A. Cuyt, P. Kuterna, B. Verdonk, and J. Vervloet. *Arithmos: a reliable integrated computational environment.* Available at http://win-www.uia.ac.be/u/cant/arithmos/index.html, 2001.

[3] D. Daney, G. Hanrot, V. Lefèvre, F. Rouiller, and Paul Zimmermann. *MPFR, The Multiple Precision Floating-Point Reliable Library.* Available at http://www.mpfr.org/, 1999.

[4] Marc Daumas and Claire Finot. Division of floating point expansions with an application to the computation of a determinant. *Journal of Universal Computer Science*, 5(6):323–338, 1999.

[5] Marc Daumas and Claire Moreau-Finot. Exponential: implementation trade-offs for hundred bit precision. In *Real Numbers and Computers*, pages 61–74, Dagstuhl, Germany, 2000.

[6] Theodorus J. Dekker. A floating point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[7] P. M. Farmwald. High bandwidth evaluation of elementary functions. In K. S. Trivedi and D. E. Atkins, editors, *Proceedings of the 5th IEEE Symposium on Computer Arithmetic.* IEEE Computer Society Press, Los Alamitos, CA, 1981.

[8] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, March 1991.

[9] J. Harrison, T. Kubaska, S. Story, and P.T.P. Tang. The computation of transcendental functions on the IA-64 architecture. *Intel Technology Journal*, Q4, 1999.

[10] D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, Reading, MA, 1973.

[11] V. Lefèvre. *Moyens arithmétiques pour un calcul fiable.* PhD thesis, École Normale Supérieure de Lyon, Lyon, France, 2000.

[12] V. Lefèvre, J.M. Muller, and A. Tisserand. Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, November 1998.

[13] J.M. Muller. *Elementary Functions, Algorithms and Implementation.* Birkhauser, Boston, 1997.

[14] D. M. Priest. Algorithms for arbitrary precision floating point arithmetic. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, pages 132–144, Grenoble, France, June 1991. IEEE Computer Society Press, Los Alamitos, CA.

[15] Jonathan R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. In *Discrete and Computational Geometry*, volume 18, pages 305–363, 1997.

[16] P. H. Sterbenz. *Floating point computation.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

[17] Shane Story and Ping Tak Peter Tang. New algorithms for improved transcendental functions on IA-64. In *Proceedings of the 14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia*, pages 4—11. IEEE Computer Society Press, April 1999.

[18] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989.

[19] P. T. P. Tang. Table lookup algorithms for elementary functions and their error analysis. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 232–236, Grenoble, France, June 1991. IEEE Computer Society Press, Los Alamitos, CA.

[20] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 18(2):211–222, June 1992.

[21] W. F. Wong and E. Goto. Fast hardware-based algorithms for elementary function computations using rectangular multipliers. *IEEE Transactions on Computers*, 43(3):278–294, March 1994.

[22] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software*, 17(3):410–423, September 1991.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399